N94-30563

6058

AIAA-94-1212-CP

A PERFORMANCE ANALYSIS METHOD FOR DISTRIBUTED REAL-TIME ROBOTIC SYSTEMS: A CASE STUDY OF REMOTE TELEOPERATION

D.R. Lefebvre and A.C. Sanderson

Electrical, Computer, and Systems Engineering Department
Rensselaer Polytechnic Institute, Troy, New York

537-63
207660
p. 11

Abstract

Robot coordination and control systems for remote teleoperation applications are by necessity implemented on distributed computers. Modeling and performance analysis of these distributed robotic systems is difficult, but important for economic system design. Performance analysis methods originally developed for conventional distributed computer systems are often unsatisfactory for evaluating real-time systems. The paper introduces a formal model of distributed robotic control systems; and a performance analysis method, based on scheduling theory, which can handle concurrent hard-real-time response specifications. Use of the method is illustrated by a case study of remote teleoperation which assesses the effect of communication delays and the allocation of robot control functions on control system hardware requirements.

1 Introduction

As ambitious robotic applications are envisioned and more complex robot designs attempted, the need increases for efficient methods to evaluate their performance. Many of these new applications will be implemented on distributed computers. For instance, remotely operated and multiple-robot applications are by their nature spatially distributed, and so necessitate a distributed real-time system for robot coordination and control. The introduction of multiple processors, communication delays, and probabilistic performance of common-access communication channels in distributed systems complicates prediction of their real-time performance.

Performance analysis methods for conventional distributed systems employ one of three approaches: simulation, stochastic models, or semantic models [7]. These methods have complementary strengths and weaknesses; so, several methods may be needed to analyze all aspects of system performance. The characteristics of these methods relevant to analyzing distributed real-time systems are summarized in Figure 1.

Simulation is arguably the most widely used approach. In a simulation model, the actual operation of the system is duplicated in software at an abstract level of detail. The fidelity of the simulation depends upon accurately representing the structural properties of the system such as precedence of operations and contention for resources; and its timing properties such as execution times, communication latencies, and sensor polling delays. A simulation can produce a full probability distribution of system response times; and so, provide a complete characterization of soft- and hard-real-time performance. Thorough characterization comes at a price: a high level of detail is needed for good accuracy, but is computationally expensive. Also, complex systems have an extremely large number of states that may necessitate excessively lengthy simulation duration to ensure that all states are exercised. For this reason, simulations are poor for proving system correctness and global properties such as boundedness and freedom from deadlock.

Stochastic models (e.g. Markov chains, queuing networks, Petri nets) are also commonly used for performance analysis, particularly for evaluating communication networks. In this approach, the system is idealized as a finite set of discrete states with known probability distributions for the transition rates between states. The model may be solved to estimate probability of each state as a function of time from which average system performance may be derived. For simple systems an efficient, analytical solution is often possible, and correctness and global properties may be determined. However, stochastic models of complex systems can be analytically intractable; requiring approximation methods which may compromise fidelity and increase computation.

[1]Copyright ©1994 by the American Institute of Aeronautics and Astronautics, Inc. All rights reserved.

Semantic modeling is a less common approach to assess system performance that arises from computational science theory of program correctness. In this approach, the logical and temporal relationship between operations of the system are defined by process algebras or assertional calculi. Correctness and timeliness properties are then established by solving the model via a theorem prover. Semantic models are effective for proving that timing specifications are satisfied, but do not necessarily provide quantitative measures of system performance. The downfall of semantic models is their computational complexity; verification is impractical for large systems.

None of the three approaches described above is fully satisfactory for estimating performance of distributed systems having hard-real-time response requirements. In a hard-real-time system, response times must never exceed specifications; and so, the system must be analyzed for worst-case performance. Simulation can produce estimates of worst-case performance, but at a high computational cost which becomes prohibitive when the system is designed for multiple concurrent responses. Stochastic models give average response times only, and thus do not provide the information necessary to gauge performance of a hard-real-time system. Semantic models excel at proving correctness and global properties, but are poor at quantifying response times. A fourth approach, based on scheduling theory, is proposed in this paper to specifically address distributed hard-real-time systems.

In the new performance analysis method, a formal model describes distributed real-time system organization and its responses to external inputs. System software is modeled as independent tasks that communicate by messages. Application of scheduling theory enables the calculation of guaranteed response times for task executions and message deliveries. The model provides a framework for formulating a constraint satisfaction problem on processor and communication channel schedules and on real-time requirements whose solution defines system response times. The performance analysis problem may be solved to minimize weighted system response time or to minimize hardware cost while meeting response time requirements. The subsequent paper sections outline the system model, show the formulation of the constraint satisfaction problem, and then illustrate its use in an example.

While this work has been motivated by the design of robot coordination and control systems, the performance analysis techniques are believed to have broader application to many distributed real-time systems.

Method	Provably Correct	Real-Time Response		Computing Expense	Limitations
		Soft	Hard		
Simulation	Poor	Good	Good	High	high level of detail needed for fidelity
Stochastic Models	Fair	Good	Fair	Moderate	some problems intractable; approx. may lead to poor fidelity
Semantic Models	Excel.	Poor	Fair	High	v. difficult to develop model; large problems intractable
Scheduling Theory	(Good)	Fair	Good	Low	pessimistic for soft-real-time;

Figure 1: Performance Analysis Method Comparison

2 Performance Analysis System Model

Distributed computer systems are composed from multiple, independent processors connected by communication links. The characteristics of distributed systems can vary widely as the result of bandwidth and propagation delay of the interprocessor connections. At the extremes are "tightly-coupled" *multiprocessor* computers in which processors share a high-speed bus, and "loosely-coupled" *multicomputer* systems which comprise separate computers connected by a network. Processor independence distinguishes distributed systems from parallel computers in which processors typically are identical, and share data streams and/or instruction decoding.

The proposed formal model can represent distributed systems with arbitrary communication network topography; and so, can model the full range from multiprocessor to multicomputer system. In fact, in the model, a single node of a multicomputer network may be a complete multiprocessor. The new method is particularly useful for loosely-coupled systems, where access to communication channels as well as processor usage must be scheduled, since few analysis techniques are available for this class of distributed system.

Because the independent computers of a distributed systems do not share physical memory, any data to be exchanged between processors must be transmitted across a communication link. The most common way to design distributed software that accommodates this restriction is to organize functions as independently-executing tasks that communicate via messages. This paradigm of tasks and messages is used in the formal model to define the system software, although the definition of a message has been generalized to include less-structured signals such as sensor inputs or control outputs.

Some tasks must execute on specific processors; for instance, a sensor polling task must run on the processor that is interfaced to the sensor hardware. However, in general, there are many choices of how to distribute software on the hardware. The actual assignment of tasks to processors has a strong influence on system performance; and so, must be specified for performance to be predicted. Optimal task assignment is an important design problem for distributed systems. We are currently experimenting with use of the new performance analysis method to guide task assignment [4].

Robotic systems, and indeed most real-time systems, interact with their environment. Sensors gather data to characterize the environment. The control system monitors sensors to detect occurrence of specific conditions or events to which the system is designed to respond. And the system effects changes to the environment through its actuators; thus forming a closed-loop system. Also, in most systems, a human operator may intervene to modify goals or to initiate actions. Performance of robotic systems may be measured in many ways: accuracy, reliability, cost, etc. As we are primarily concerned with the computer system providing robot coordination and control, performance will be defined as the end-to-end response time from when a condition can be sensed until a control signal is sent to actuators. Therefore, system response requirements are identified by input-output events and a response time specification. The requirement specifies a maximum response time since we are dealing with *hard* real-time systems.

From this description we see that four components are needed to fully describe a distributed real-time system:

- software system model
- hardware system model
- assignment of tasks to processors
- system response requirements

In the definition of each model component, covered in the following subsections, we have attempted to describe distributed real-time systems in terms that are

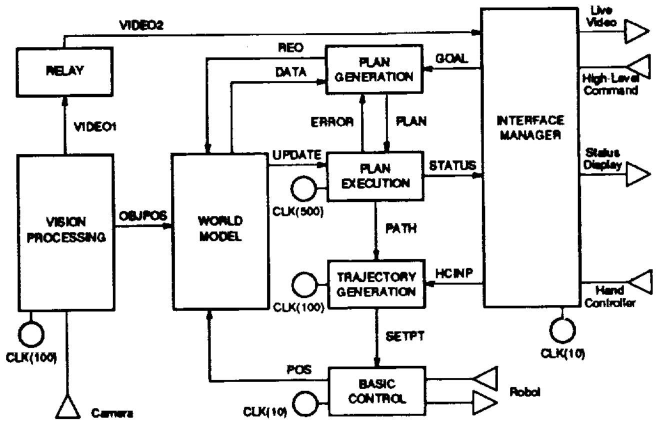

Figure 2: Software Model of Teleoperation Example

as similar as possible to how they are actually constructed. While this tends to specialize the model, it has the benefit of providing a more natural representation of a system implementation which, hopefully, improves ease of use and accuracy.

2.1 Software System Model

A distributed robotic application is typically constructed from many, concurrent tasks that execute on multiple processors, and communicate by message passing between tasks, or between task and sensor or actuator. Each task corresponds to a software module, and the resulting system may be described by a directed graph with nodes corresponding to tasks and arcs representing messages. Each task is a separate software module that may execute periodically or upon demand ("aperiodic" or "event-driven"). This *system level* graph defines the topography of the communications between tasks.

Figure 2 shows a system level view of a simplified control system for the teleoperated robot example that will be described in Section 4. The example employs a hierarchical architecture loosely modeled on the NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM) [1]. System software is modeled with five periodic tasks and three event-driven tasks, which are shown as boxes in the figure. Periodic tasks are identified by their clock inputs (circles). Input (sensors, keyboards) and output (actuators, displays) devices are represented by triangles whose orientation denote direction of data flow. Messages are shown as arrows from sending task to receiving task. A total of fourteen messages are transmitted between tasks in the example.

At a more detailed *module level*, each task is viewed as a finite state machine where task states or actions are nodes, and transitions between actions are directed arcs. The transition arcs are labeled with Real-Time Logic predicates [3] which define the condition causing the transition to occur. The purpose of the module level view is to define the response of an individual task to the input messages it receives. The finite state machine representation of the task allows a different computation time and different set of output messages to be defined for each input message.

Each action node in the finite state machine represents a deterministic sequence of operations that are delimited by a decision branch or a message transmission/arrival. When a node is entered it executes for a fixed time interval and then optionally sends a message prior to blocking in that state or transitioning to another. Computation times are associated with actions, while transitions are instantaneous. The optional messages produced by module actions correspond to the messages output from modules at the system level. Messages are identified only by type and bit length; the data values contained in a message are not considered.

Figure 3 shows the finite state machines for two tasks from the teleoperated robot example. The VISION PROCESSING task periodically acquires a camera image frame, transmits the frame as a VIDEO1 message, processes the image to locate objects in the robot's environment, and then outputs the positions of the objects in a OBJPOS message. Task processing is initiated when a CLK signal is received; and, when complete, the task returns to Idle Wait state to await the next signal. Figure 3b shows the finite state machine for the aperiodic PLAN GENERATION task. This task is invoked by the arrival of a message rather that a clock signal; and contains two paths so that GOAL and ERROR messages may be processed differently. Note that execution of the task is interrupted at Action 4 while it waits for requested data. Definition of periodic and aperiodic tasks are essentially the same at the module level — differing only by whether a clock signal or a message activates the task.

2.2 Hardware System Model

The purpose of the hardware system model is to describe how processors are interconnected, and the capabilities of processors and communication channels. The principal capabilities of interest are processor

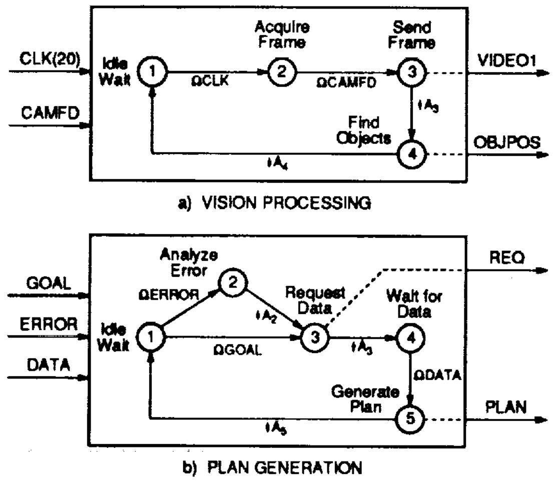

Figure 3: Finite State Machines for Example Modules

speeds, and communications bandwidth and propagation delay.

Processor interconnections are represented by a *hardware graph* in which graph nodes correspond to processors, and graph arcs indicate the one-hop communication links between processors. Dedicated, unidirectional communication channels are shown as directed arcs; and shared communication channels (half-duplex or broadcast media) as sets of non-directed arcs. Any connection topography can be modeled in this way including loosely-coupled multicomputer networks, bus-based multiprocessors, and combinations of the two.

Figure 4a is a diagram of a multicomputer system with four single-processor workstations and a 4-processor multiprocessor connected by a local area network. One sensor and one actuator are interfaced to the multiprocessor. Figure 4b is the corresponding hardware graph. Note how the shared multicomputer LAN and the shared multiprocessor bus are expanded into sets of bi-directional links that fully interconnect all processors sharing each communication medium. There are no dedicated links in this example.

2.3 Task Assignment

The distribution of software modules onto computer hardware is described by first numbering all tasks and processors, and then defining an *assignment function* α which maps a task to a processor. Thus if task t_i is assigned to processor p_j then $\alpha(i) = j$. This definition allows us to reference the hardware properties of the processor on which a task runs.

A similar assignment function can be defined to ref-

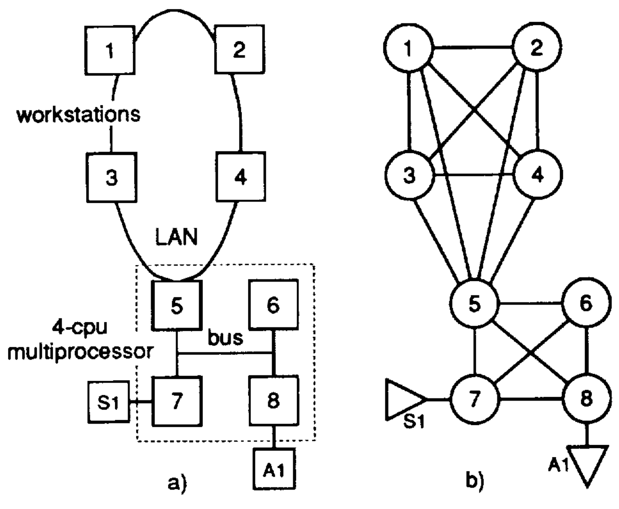

Figure 4: Hardware System Model Example

erence the communication hardware used by a message; thus if message m_i is assigned to communication link l_j then $\alpha(i) = j$. Once tasks are assigned to processors the communication link over which a message travels is defined. Therefore the message assignment function can be derived from the task assignment function plus the software and hardware system models; and so, we only need to specify the task assignment function.

2.4 Response Requirements

For this work, the principal performance measure is response time. System response time is defined as the time interval between occurrence of an external event and the system response. When sensor polling delays and actuator response times are factored out, the response time can be expressed as the time between an external input (sensor reading, operator command, etc.) and an external output (control signal, display update, etc.) of the control system.

System response requirements specify the events to which the system must respond, the expected actions, and response time. Requirements correspond to the environmental constraints on the robotic system. We will consider only hard-real-time requirements in which a maximum response time is specified and must be satisfied.

Most robotic systems will respond to many different events; and so, multiple response requirements will be defined. In hard-real-time applications, the system is expected to process concurrent events within their maximum response times under all load conditions. It is this requirement for concurrent responses that makes analysis of hard-real-time systems difficult. Contention for processors and communication channels will vary as the mix of concurrent events and their relative overlap varies. For instance, it is difficult to ensure that sufficient simulations have been performed such that the worst-case combination of concurrent events is modeled. And, average response times obtained from stochastic models provide no information regarding response degradation under load. A key advantage of a scheduling theory-based approach is that its results hold for all phasings of task invocations, i.e. degree of overlap of concurrent events.

3 Formulation of Performance Analysis Constraint Satisfaction Problem

With the information contained in the system model described above, a constraint satisfaction problem can be formulated whose solution yields an estimate of system response times. Performance of the distributed robotic system is defined by a set of constraint equations relating hardware, software, and response times. These equations are presented in the following subsections.

This system of equations is underconstrained; and so, a cost function is added to reduce the degrees of freedom. Different solution objectives can be achieved with different cost functions. In particular, the constraint equations may be solved to yield minimum system response times for fixed hardware capacities, or to find minimum-cost hardware which can meet all system response time requirements.

The problem is summarized as:

- Minimize system response times <u>or</u> hardware cost
- Subject to:
    1. Having a feasible schedule on every processor and communication channel
    2. Meeting system response time requirements
    3. Satisfying bounds on individual task and message response times

3.1 Cost Functions

If no constraints are mutually exclusive then the constraint satisfaction problem can be solved. However, since it is typically underconstrained, the problem can have an infinite number of solutions. A cost function is included which introduces additional constraints to ensure that only one solution is produced.

Through our choice of cost function we can direct the solution to achieve various design objectives.

System response time is one possible cost function. Since the system may have multiple responses, a weighted sum of response times is used to give a single-valued function. This allows us to emphasize one system response over another. A large penalty is assigned for exceeding a system response requirement, so all requirements are met before responses are further minimized. When this cost function is used, hardware capacities are held constant; hence, this form is useful for evaluating existing hardware.

As an alternative, hardware capacities may be allowed to vary, and hardware cost used as the cost function. The problem solution yields values for hardware capacities as well as system response times. This form of the constraint satisfaction problem is useful for evaluating proposed hardware designs. The example performance analysis in Section 4 is formulated in this manner.

3.2 Scheduling Constraints

A principal distinction between performance analysis methods is how they handles resource contention. Analysis methods for real-time systems must be able to represent the order of internal system events so that resource contention can be modeled. Usually this means that the protocols for scheduling task executions and message deliveries must be known. Simulation methods use this information directly; while stochastic models represent resource contention probabilistically. The scheduling-based performance analysis method presented here requires that a priority-based, preemptive scheduling protocol be employed for both processors and communication channels. Real-time operating systems typically implement such protocols for processors; however, communication protocols supporting time-constrained messages are recent developments [9][2], and are much less common.

The reason the scheduling-based method is restricted to priority-based, preemptive protocols is that it depends on their predictable properties. With this class of scheduling protocol the execution time of the highest priority task is always known, and the worst-case execution times of lower priority tasks can be determined by assuming all higher priority tasks must execute first. In 1973, Liu and Layland [6] proved several properties of priority-based, preemptive scheduling protocols and introduced an analysis technique known as the *rate monotonic scheduling algorithm*. They established criteria for multiple tasks executing periodically on a single processor that, when satisfied, guarantee a schedule can be found in which all tasks meet their execution deadlines. They also showed that an optimal schedule is obtained by assigning priorities based on task periods — shortest period task has highest priority. The original work on scheduling uniprocessors has been extended to systems with aperiodic tasks and to shared communication channels, and is now referred to as generalized rate monotonic scheduling [5][8].

In the proposed performance analysis method, scheduling theory criteria are used to identify the conditions under which a set of tasks [messages] can be scheduled such that they are guaranteed to meet execution [delivery] deadlines. These deadlines are then used as guaranteed response times. We have developed a modified form of the generalized rate monotonic scheduling algorithm which applies to the robotic system model with event-driven tasks and real-time constraints.[2] This modified scheduling criterion gives the minimum speed S^* of a processor [or communication link] required to successfully schedule the tasks [or messages] assigned to it:

$$S^*_j(\bar{C},\bar{r},\bar{\theta}) = \max_{\{1\le i\le N_t\}} \min_{\{\tau\in SP_i\}} \left(\sum_{n=1}^{i} C_n \left\lceil \frac{\tau}{\theta_n} \right\rceil \right) / \tau \tag{1}$$

where $\bar{C}$, $\bar{r}$, and $\bar{\theta}$ are vectors of computation times, deadlines (guaranteed response times), and periods of tasks [messages] assigned to processor [link] j, respectively. N_t is the number of assigned tasks [messages], and SP_i is the set of critical scheduling points as defined by:

$$\begin{aligned} SP_i &= \{(k-1)\theta_j + r_j \mid {\scriptstyle j=1,\dots,i;\, k=1,\dots,\lfloor \frac{r_i+\theta_j-r_j}{\theta_j} \rfloor}\} \\ &\quad \bigcup \{k \cdot r_j \mid {\scriptstyle j=1,\dots,i;\, k=1,\dots,\lfloor \frac{r_i}{\theta_j} \rfloor}\} \end{aligned}$$

Note that computation times C_i are normalized for a "standard" processor defined to have a *relative speed* of 1. Processor speed and S^* are expressed as relative speeds by ratioing to the standard processor. Messages and communication channels are treated in the same manner.

[2]Strictly speaking, since the technique uses deadlines rather than periods it should be referred to as deadline-monotonic scheduling. However, for clarity the more familiar term will be used.

The scheduling constraints require that the minimum relative speed S^* for a feasible schedule be less than or equal to the relative speed S of the processor or communication link:

$$S_j^*(\bar{C},\bar{r},\bar{\theta}) \leq S_j \quad \text{for every processor and link } j \quad (2)$$

The scheduling criterion defined by equation 1 essentially forms a ratio between demand for execution capacity (summation term) and available capacity (τ). The ratio is checked at critical scheduling points which occur at deadlines and periods. Execution capacity demand is calculated for all tasks of priority i and higher priority tasks which may preempt it. The minimum ratio over all scheduling points reflects the lowest speed at which these tasks are schedulable for a given priority. Finally, the ratio is checked for all priorities, and the worst case defines the relative speed needed to successfully schedule the assigned tasks or messages.

3.3 System Response Time Constraints

As defined in Section 2.4, system response requirements are specified in terms of the external event which invokes a response, the expected system action, and response time. An external event detected by the system's sensors will trigger a cascade of task executions and message transmissions. Many tasks may execute concurrently and multiple messages may contend for shared communication channels. The precedence of task executions and message transmissions associated with a particular event can be derived from the software model and is represented as a weighted directed acyclic graph called an *event response graph*. Graph arcs are weighted with task execution times and message delivery times, which are dependent on the underlying hardware capabilities. Since the graph is deterministic, a critical path through the graph can be found that defines system response time for the specific event.

As an example, consider the response of the system from Figure 2 to a high-level command input by an operator. The command is received by the INTERFACE MANAGER task which interprets the command and then transmits a GOAL message to PLAN GENERATION. In subsequent processing steps data is obtained from the WORLD MODEL, a plan created and sent to PLAN EXECUTION, and so on until the system response to the high-level command is produced at the robot. The complete sequence of task executions and message transmissions is shown in the event response graph in

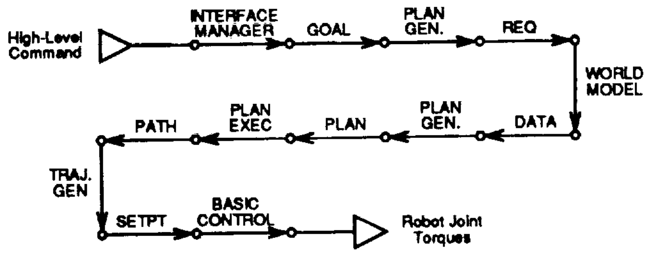

Figure 5: An Event Response Graph

Figure 5. This simple example has a linear critical path; but, in general, the critical path may contain parallel legs. Control system response time is calculated by summing guaranteed task execution times of the seven tasks in the graph including polling delays at the periodic tasks, plus guaranteed message delivery times of the six messages including propagation and switching delays, plus communication time associated with sensor input or actuator output. Note that the PLAN GENERATION task appears twice in the example event response graph. The first invocation of PLAN GENERATION is in response to a GOAL message, and the second in response to a DATA message. Execution times for PLAN GENERATION are different in each instance as defined in the module level finite state machine description of the task (see Section 2.1).

The fact that event response graphs must be deterministic does not prevent us from modeling probabilistic events such as failures. In these cases, an event response graph would be developed to represent the processing that occurs for each possible outcome; and, potentially, each outcome could have a separate hard-real-time response requirement. If a system is required to meet a response time specification even in the presence of failure then only the more restrictive situation would have to be modeled — probably the case including the additional processing to accommodate failure. Alternatively, a less demanding response time requirement could be defined for failure processing which would yield a less costly control system design. This type of analysis enables us to study tradeoffs between system reliability and cost.

An event response graph is constructed for each system response having a time requirement. Since guaranteed task execution times and guaranteed message delivery times are solution variables of the problem, system response time can be determined by summing the variables corresponding to the weights on the event response graph. The constraint equations are formed by requiring that system response time must be less than its requirement for each response:

$$\sum_{t_i,m_i \in CP_k} r_i \leq R_k \quad \text{for all responses k} \qquad (3)$$

where r_i is the guaranteed response time of task or message i, CP_k is the set of tasks and messages on the critical path for response k, and R_k is the system response time requirement.

3.4 Task/Message Response Time Bounds

Response time for an individual task or message is bounded. Response time can not be less than the time required to execute the task or transmit the message, and is not allowed to be greater than its period. This upper bound results from a restriction that at most one invocation of a task is allowed to execute at a time. For aperiodic tasks, a parameter analogous to period is specified to be the minimum interval between executions. These bounds place the following constraints on guaranteed response times:

$$\frac{C_i}{S_{\alpha(i)}} \leq r_i \leq \theta_i \quad \text{for all tasks and messages} \qquad (4)$$

where $S_{\alpha(i)}$ is the relative speed of the processor to which t_i or m_i is assigned (recall that α is the assignment function), θ_i is the period or minimum interarrival time of the task or message, and the other terms retain their earlier definitions.

Task/message response time bounds can be represented as simple variable bounds for constraint satisfaction problems with constant processor and communication channel speeds since all of the terms in the calculation of the lower and upper bounds would be known and constant. However, if hardware speeds are solution variables, then the lower bounds must be incorporated as nonlinear constraint equations.

3.5 Solving Constraint Equations

In summary, to analyze the performance of a distributed robotic system we first define the system by the model outlined in Section 2; then form the system of constraints from equations 2, 3, and 4. The constraint satisfaction problem is solved to minimize the cost function, i.e. to minimize weighted system response time, or to minimize hardware cost. The solution provides times for all system responses, guaranteed response times for task executions and message deliveries, and processor and communication channel speeds.

A nonlinear programming method is needed to solve the constraint equations. Unfortunately, although equation 1 is continuous it is not smooth. Therefore, nonlinear methods such as sequential quadratic programming and others that require smooth derivative information can not be used. The system of constraints has been successfully solved with a successive linear programming approach. We believe that this approach works because the partial derivatives of equation 1 are piecewise-linear.

4 Example Use of Analysis Method

This section presents an example use of the new performance analysis method for design of the control computer system of a teleoperated robot. The minimum-cost hardware formulation will be used to select capacities of processors and communication links for various design conditions of communication delay and task assignment.

Control software is organized in a "NASREM-like" architecture as seen earlier in Figure 2. The standard components of sensory processing, world modeling, task decomposition, and operator interface are all included; however, only the task decomposition functions are modeled in sufficient detail to show a hierarchical organization. The eight tasks comprising the system are listed in Table 1 with their relative computation times and execution periods. Note that the task decomposition functions of PLAN GENERATION, PLAN EXECUTION, TRAJECTORY GENERATION, and BASIC CONTROL form a hierarchy with execution period differing by an order of magnitude between levels. Parameters for the messages transmitted among the tasks are listed in Table 2.

Task	Comp Time, ms	Period, ms
Basic Control	4	10
Traj. Generation	30	100
Plan Execution	50	1000
Plan Generation	2000	-
World Model	50	-
Vision Processing	170	100
Video Relay	0.1	100
Interface Manager	10	10

Table 1: Task Parameters for Example

Figure 6 shows the control system hardware for the teleoperation example. It includes a *local* processor at ground station, a *remote* processor at an or-

Message	Length, kbits	Period, ms
GOAL	7.8	-
PLAN	7.8	-
PATH	3.7	1000
HCINP	0.4	10
SETPT	1.1	100
POS	0.4	10
OBJPOS	7.3	100
UPDATE	7.8	100
STATUS	7.8	1000
ERROR	0.1	-
REQ	0.8	-
DATA	7.8	-
VIDEO1	25	100
VIDEO2	25	100

Table 2: Message Parameters for Example

bital facility, and *control* and *vidpp* processors on the robot to support dedicated control and video preprocessing functions. Three communication channels connect these processors: unidirectional *uplink* and *dnlink* channels between ground and orbit, and a radio network, designated *rnet*, for communications between robot processors and the orbital facility. The nominal assignment of tasks to processors locates VISION PROCESSING on *vidpp*, BASIC CONTROL on *control*, INTERFACE MANAGER on *local*, and all remaining tasks on the *remote* processor.

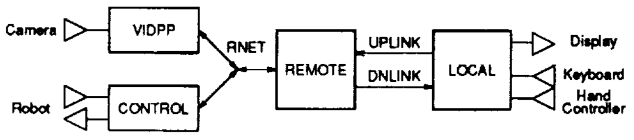

Figure 6: Hardware for Teleoperation Example

Five time-critical responses are specified, and serve as the hard-real-time system response time requirements. They are listed on Table 3. The control system must display information about the work site in three forms: live video at 10 frames/second, a reconstruction of the world model updated by object recognition, and a model showing robot position. The system must guarantee that data from each of the three sources is delivered to the INTERFACE MANAGER in 2.4 seconds (2400 ms) so that it can be fused into a consistent display. An operator controls the robot either indirectly through high-level commands, or directly via a hand controller. The system is expected to respond to high-level commands in 9600 ms, and hand controller input in 1200 ms. As covered in section 3.3, an event response graph is constructed for each system response requirement to identify the tasks and messages invoked to process each response.

Description	Mnemonic	Requirement
Display Live Video	LIVE_DSP	2400 ms
Display World Model	WM_DSP	2400 ms
Display Robot Position	ROB_DSP	2400 ms
Respond to HL Command	CMD_RSP	9600 ms
Respond to Hand Controller	HC_RSP	1200 ms

Table 3: System Response Requirements for Example

Relative costs for processors and communication channels were modeled with a power function: $cost = multiplier \times speed^{exponent}$. Ground facilities were assigned a 1.0 multiplier, orbital facilities were assumed to have an order of magnitude higher multiplier, and processors on the robot have an additional factor of two premium. Exponents of 2.0 for ground and 1.5 for orbital facilities were used. The cost model should have an additional additive factor; but the SLP solution method being used cannot support it.

4.1 Effect of Communication Delays

In the first design study, the effect of communication delay on processor and communication channel capacity is examined. The new performance analysis method is used to find the minimum hardware needed to guarantee that system response time requirements are achieved. Communication delays of 100, 500, and 1000 ms are studied. These values represent propagation and switching delays only; and so, may appear low compared to customarily quoted values which include scheduling delays due to traffic contention. The performance analysis method computes the scheduling delays.

Table 4 summarizes key results. As required, all system responses meet requirements. At 100 and 500 ms delays the high-level command response is limiting; whereas, the world model display response limits at 1000 ms delay.

Most non-limiting responses differ between cases by an amount equal to the difference in communication delay. This is a consequence of the solution method which focuses on the requirements that constrain hardware speed while essentially ignoring responses not at a limit. Requirements for non-limiting responses could be lowered to the reported values without affecting hardware speeds.

Faster processors and communication channels are needed as communication delay increases. At delays of

500 ms and lower, the more expensive *control* and *vidpp* processors are at minimum capacity needed to meet execution periods of their assigned tasks. A modest increase in *remote* processor speed is sufficient to accommodate a communication delay of 500 vs. 100 ms. However, for the 1000 ms delay, all processor speeds must be higher in order to meet system response requirements.

Total relative hardware cost differs little between 100 and 500 ms cases. However, the cost of the video preprocessor dominates total cost, thereby masking the 10% difference in cost of all other components between the cases. The design for 1000 ms delay is significantly more costly: 246% total and 137% system excluding *vidpp* costs relative to the 500 ms design.

The point of this analysis is not to draw conclusions regarding an admittedly over-simplified teleoperated robot application, but rather to demonstrate a possible use for the new performance analysis method. It is feasible to guide key design decisions, in this case by examining tradeoffs between control system costs and communication switching infrastructure, through use of real-time system analysis.

Case #	1	2	3
Comm Delays, ms	**100**	**500**	**1000**
System Responses, ms			
- LIVE_DSP	360	760	1210
- WM_DSP	1760	2060	**2400**
- ROB_DSP	1580	1880	2270
- CMD_RSP	**9600**	**9600**	9380
- HC_RSP	280	670	1160
Processor Capacity			
- control	0.40	0.40	0.65
- vidpp	1.70	1.70	3.55
- remote	1.22	1.36	1.52
- local	1.00	1.00	1.12
Comm Link Capacity			
- uplink	0.99	0.98	1.03
- dnlink	0.99	0.98	0.99
- rnet	0.10	0.11	0.11
Relative Cost			
- system ex vidpp	0.90	1.00	1.37
- vidpp	1.00	1.00	3.02
- total system	0.97	1.00	2.46

Table 4: Effect of Communication Delays

4.2 Effect of Task Assignment

Another use of the new performance analysis method is illustrated in this section as a design study evaluating the effect of task assignment on hardware cost. Communication delays are fixed at 500 ms for all cases. In the base case, PLAN GENERATION, PLAN EXECUTION, and TRAJECTORY GENERATION tasks execute on the *remote* processor, and the BASIC CONTROL task on the *control* processor.

The effect of moving first PLAN GENERATION and then PLAN EXECUTION to the *local* processor was modeled with the results shown in Table 5. Cost savings can be obtained by shifting computing load from expensive orbital processors to lower cost ground computers while still meeting response time requirements. For this simplified example, the analysis suggests that the savings may be substantial, and may motivate further study to assess the impact on other mission-critical factors such as the reliability and safety implications of remote computing.

Migrating dedicated processing at the robot to the somewhat less expensive computing available at Space Station may also be cost effective. Table 6 summarizes modeling results for moving the CONTROL task to the *remote* processor. This reassignment eliminates the *control* processor which is replaced by simpler hardware to receive control signals from *rnet*; and *remote* processor capacity is correspondingly increased. Communication latency of the control signals are on the order of 0.3–0.4 ms which should be acceptable. The full benefit of relocating control functions may not be achievable since some at-robot processing capability is required for safety functions which have not been modeled.

5 Future Work

Currently we are working to improving the efficiency of the performance analysis method; in particular, to increase robustness of the successive LP solution approach and to decrease computation time. The principal motivation for improving solution efficiency is so that the performance analysis method may be embedded in a genetic algorithm with the objective of finding near-optimal task assignments. If successful, this would provide a powerful tool for designing distributed real-time systems in which software module allocations and hardware are optimized concurrently.

Other activities are aimed at demonstrating the capabilities of the performance analysis method on a variety of robotic systems, and directly comparing results to those obtained from simulation and stochastic models. Theoretical and experimental verification of performance analysis tools will provide an important contribution to the field of robotics, and will form the

Case #	2	4	5
Comm Delays, ms	500	500	500
Task Assignments			
- Plan Gen.	remote	local	local
- Plan Exec.	remote	remote	local
- Traj. Gen	remote	remote	remote
- Control	control	control	control
System Responses, ms			
- LIVE_DSP	760	820	900
- WM_DSP	2060	**2400**	1970
- ROB_DSP	1880	2220	1790
- CMD_RSP	**9600**	**9600**	**9600**
- HC_RSP	670	670	680
Processor Capacity			
- control	0.40	0.40	0.40
- vidpp	1.70	1.70	1.70
- remote	1.36	0.92	0.81
- local	1.00	1.32	1.35
Comm Link Capacity			
- uplink	0.98	0.92	0.63
- dnlink	0.98	0.90	0.21
- rnet	0.11	0.11	0.08
Relative Cost			
- system ex vidpp	1.00	0.72	0.62

Table 5: Effect of Shifting Tasks to Local Proc

Case #	4	6	5	7
Comm Delays, ms	500	500	500	500
Task Assignments				
- Plan Gen.	local	local	local	local
- Plan Exec.	remote	remote	local	local
- Traj. Gen	remote	remote	remote	remote
- Control	control	**remote**	control	**remote**
System Responses, ms				
- LIVE_DSP	820	800	900	820
- WM_DSP	**2400**	2360	1970	2060
- ROB_DSP	2220	2170	1790	1870
- CMD_RSP	**9600**	**9600**	**9600**	**9600**
- HC_RSP	670	680	680	690
Processor Capacity				
- control	0.40	0	0.40	0
- vidpp	1.70	1.70	1.70	1.70
- remote	0.92	1.14	0.81	0.96
- local	1.32	1.32	1.35	1.36
Comm Link Capacity				
- uplink	0.92	0.92	0.63	0.84
- dnlink	0.90	0.93	0.20	0.60
- rnet	0.11	0.06	0.08	0.03
Relative Cost				
- system ex vidpp	0.72	0.66	0.62	0.51

Table 6: Effect of Shifting Control Task

basis for more efficient development of new robotics applications in the future.

Acknowledgment

This work has been conducted at the Center for Intelligent Robotic Systems for Space Exploration (CIRSSE) with support from NASA Grants NGT-10000 and NAGW-1333.

References

[1] J. S. Albus, H. G. McCain, and R. Lumia. NASA/NBS standard reference model for telerobot control system architecture (NASREM). Technical Report NIST Technical Note 1235, National Institute of Standards and Technology, Gaithersburg, MD, April 1989.

[2] B. Chen, G. Agrawal, and W. Zhao. Optimal synchronous capacity allocation for real-time communications with the timed token protocol. In *Proc. IEEE Real-Time Systems Symposium*, pages 198–207, Pheonix,AZ, 1992.

[3] F. Jahanian and A. K. Mok. Safety analysis of timing properties in real-time systems. *IEEE Trans. on Software Engineering*, 12(9):890–904, September 1986.

[4] D.R. Lefebvre and A.C. Sanderson. Performance evaluation and task assignment in distributed robotic systems. In *Proc. IEEE Conf. on Robotics & Automation*, San Diego, May 1994 (submitted).

[5] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *Proc. IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[6] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of The Association for Computing Machinery*, 20(1):46-61, January 1973.

[7] A. A. Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems.* MIT Press, 1986.

[8] L. Sha and S.S. Sathaye. A systematic approach to designing distributed real-time systems. *IEEE Computer*, 26(9):68-78, September 1993.

[9] W. Zhao, J. A. Stankovic, and K. Ramamritham. A window protocol for transmission of time constrained messages. *IEEE Trans. on Computers*, 39(9):1186-1203, September 1990.